Quantifiers and n-ary Binders: an OpenMath Standard Enhancement Proposal

Lars Hellström

Department of Mathematics and Mathematical Statistics, Umeå University, Umeå, Sweden;
lars.hellstrom@residenset.net

Abstract

It is proposed that the restriction in the OpenMath standard that an OMBIND element must have exactly three children should be lifted, to support more general binder symbols. The case of logics with generalised quantifiers is described in some detail, since these turn out to not have a natural encoding within OpenMath 2.0, because of precisely this restriction. That restricting quantifiers to a single body should have such consequences is not trivial, but follows from a theorem in the Logic branch of Philosophy.

1 Introduction

In the OpenMath standard, there are four kinds of non-basic OpenMath objects: application (OMA), attribution (OMATTR), binding (OMBIND), and error (OME). Errors clearly have a special standing as denoting an exception state, and attributions are highly fluid in that flattening or unflattening a sequence of attributions has no effect on its semantics. This leaves the job of encoding the structure of a formula to application and binding objects: application is the main workhorse, whereas binding objects are built where a variable is being bound.

OMA and OMBIND have considerable similarities. Both have as first child a 'head' object (often a dedicated symbol) that determines the detail semantics of the construction as a whole; remaining children may typically be though of as something upon which the head acts. For several mathematical concepts, such as integrals, sums, and limits, formalising one as a binder acting upon an expression or as an application acting upon a function may be equally natural choices. Thorough textbooks defining integrals may for example [9] start with defining 'definite integral $\int_a^b f$ of a function' using upper and lower sums, and only when that whole machinery is in place switch to the more colloquial 'definite integral $\int_a^b f(x)\,dx$ of an expression'; the former is an application, the latter is a binder, but the mathematical content is the same.

There is however one factor in the OpenMath standard that prevents viewing an OMBIND simply as an OMA beefed-up with the additional power to bind variables, namely that an OMA can have any number of children, whereas an OMBIND must have exactly three: the head, an OMBVAR containing the variables to be bound, and one body object; a structural analogy with OMA would rather require that there could be zero or more body objects. Not having that freedom becomes a noticeable restriction quite soon beyond the most elementary integral and sum concepts, since it is quite common in calculations that these carry conditions (involving the bound variables) which are as important as the integrand itself for how one should proceed. These conditions are not natively part of the integrand (even if it can be useful to introduce devices that let one so incorporate them [7]), so they should intuitively appear as a separate child of the OMBIND.

Not surprisingly, this problem has been noticed and discussed before [5], in the context of aligning OpenMath and MathML, since MathML 2 allowed dedicated `<condition>` elements on integrals. The outcome of this alignment was however *not* to enhance OpenMath, but rather to

semi-deprecate the likes of `<condition>` by leaving them out of Strict Content MathML, instead prescribing for them a rewrite-to-SCMML procedure by way of `<domainofapplication>` and `set1#suchthat` that thereby becomes the normative definition of their semantics. In other words, a view got to reign that the addition of a condition clause in a binding object does not represent a switch to a different but related symbol (such as between `arith1#minus` and `arith1#unary_minus`), but rather a streamlined presentation of a more complex formula (the condition really figures in the calculation of a restricted domain of application).

This view represents a feasible way of formalising concepts such as sums with conditions, but whether it also provides a faithful representation of these symbols with respect to the practices for how they are manipulated is quite a different matter. In for example

$$\sum_{i=0}^{n}\sum_{k=i}^{n}\binom{k}{i}x^i y^k = \sum_{\substack{i,k\in\mathbb{Z}\\0\leqslant i\leqslant k\leqslant n}}\binom{k}{i}x^i y^k = \sum_{k=0}^{n}\sum_{i=0}^{k}\binom{k}{i}x^i y^k = \sum_{k=0}^{n}(x+1)^k y^k \tag{1}$$

the key idea is that the left double sum mechanically translates to a combined sum with condition $0 \leqslant i \leqslant n \ \wedge \ i \leqslant k \leqslant n$ whereas mechanical translation of the right double sum yields the condition $0 \leqslant k \leqslant n \ \wedge \ 0 \leqslant i \leqslant k$—both of which are neatly summarised as $0 \leqslant i \leqslant k \leqslant n$. With the view that the condition sum is *technically*

$$\sum_{(i,k)\in\{(i,k)\in\mathbb{Z}^2\,|\,0\leqslant i\leqslant k\leqslant n\}}\binom{k}{i}x^i y^k$$

or even

$$\texttt{arith1\#sum}\Big(\texttt{set1\#suchthat}\big(\mathbb{Z}^2, \lambda i,k.\,(0\leqslant i\leqslant k\leqslant n)\big), \lambda i,k.\,\tbinom{k}{i}x^i y^k\Big) \tag{2}$$

the step grows longer, mostly because it becomes necessary to reverse the introduced ordering of the bound variables. It likewise becomes much less straightforward to argue that a summand may be rewritten subject to some piece of information expressed in a condition when that condition is two steps further away from the sum.

Still, it will likely remain dependent on point of view whether the conditioned sum in (1) *is* technically an expression of the form (2) or rather just something *equivalent to* it. Because K–14 mathematics undeniably provides for these kinds of binding-reduced presentations, it is hard to argue within that context that a more general OMBIND is *necessary*. Therefore, if one wishes to drive home such a point of necessity, one should rather look beyond the K–14 horizon, and in particular at contexts that do not have set theory and lambda calculus built in. Such contexts exist, and one may even add the prerequisite that they have practical applications, but they are probably not what one would call "part of mainstream mathematics", since the relevant literature tends to rather be classified as computer science or philosophy. (But then again, so is much of lambda calculus and set theory.)

Next, Section 2 gives an introduction to a body of theories where generalised binders are common and important. Section 3 then details the enhancements of the OpenMath standard here proposed, and finally Section 4 discusses some objections that have been raised in the past.

2 General quantifiers

Quantifiers are concepts of which different mathematicians seem to have unusually varied understandings. A very striking example of this is provided by the *Encyclopaedia of Mathematics*

entry [10] on the subject, where the original entry basically states that 'quantifier' is the common term for $\forall$ and $\exists$, but the supplementary comment describes the subject as 'far more involved than suggested above'; a nice overview of what it contains is given in [11]. What can currently be found in the official OpenMath content dictionaries does not go beyond the former view, but the standard as such should be able to encode also everything brought up in the latter.

Part of the difference in views may be due to that the contexts in which one tends to encounter quantifiers other than the basic $\forall$ and $\exists$ (and "shorthand" variants of these, such as the unique existence quantifier $\exists!$) are not that of mathematics in general, but rather some specialised subsets of mathematics. The best-known example of such a subset is probably the use of Monadic Second-Order (MSO) logic within formal language theory, where several classes of languages have been characterised using MSO logic (see for example [1, 2]). The idea here is that every element of a formal language (e.g. every string, if considering word languages) is taken as defining a model for some logic, and a typical theorem might be that a language belongs to a particular class if and only if there is some formula in the logic under consideration which is true for precisely those models which correspond to elements of the language.

In second-order logic, one may quantify over predicates ("relation symbols") as well as variables, but the 'monadic' imposes the restriction that the predicates being quantified may only have one argument. Since a relation with one argument is essentially a set ($P(a) \iff a \in P$), this means MSO logic gives one the power to quantify over sets of atoms (but not over for example sets of pairs of atoms) *without introducing any primitives of set theory* as such, since that would let the logic express far too much for these purposes. The point is not to build a formal theory and start deriving theorems stated in MSO logic, but to use that logic to describe classes of models. For suitably weak logics, truth may well be an algorithmically decidable property.

Quite a lot can be done in MSO logic with just the (possibly second order) universal and existential quantifiers, but the fact that these bind "set variables" suggests a range of additional conditions that can be imposed. One might quantify with respect to the finite sets, the sets of even cardinality, or the sets of prime cardinality. Having the same quantifier bind several variables, one can quantify with respect to pairs of sets of equal cardinality. None of these things are beyond what OpenMath 2 supports, but they are mentioned here to give a hint of how useful expressive power can be built into the quantifiers of a language, rather than provided via predicates or functions.

What OpenMath 2 does not support is *Lindström quantifiers*,[1] since these bind variables in several subformulae simultaneously; if Q_τ is a Lindström quantifier and $\varphi_1(x), \ldots, \varphi_k(x)$ are well-formed formulae, then

$$Q_\tau x\big[\varphi_1(x), \ldots, \varphi_k(x)\big] \tag{3}$$

is a well-formed formula in which there are no free occurrences of x. The natural encoding of this formula in OpenMath (XML) would be

```
<OMBIND>
  ⟨encoding of Q_τ⟩
  <OMBVAR>
    <OMV name="x"/>
```

[1] There seems to be some disagreement about terminology here. In [11], the Q_τ of formula (3) would be called 'a generalised quantifier of type $\langle 1, \ldots, 1\rangle$', whereas the term 'Lindström' in addition is taken to suggest that several variables are being bound simultaneously (polyadic quantifier). Other authors seem to use it regardless of the number of variables bound. For OpenMath, it is rather the number of subformulae that the quantifier combines that is the novelty.

```
  </OMBVAR>
  ⟨encoding of φ1(x)⟩
     ⋮
  ⟨encoding of φk(x)⟩
</OMBIND>
```

but this is currently not allowed, since the number of children of the `<OMBIND>` is $2 + k$ rather than exactly 3.

The matter of how an interpretation is assigned to such a quantifier Q_τ is admittedly somewhat involved; for a fully rigorous definition see [8, Ch. 5], [6, Ch. 12], or [11]. What one must supply is a rule which assigns a truth value to the satisfaction relation

$$(\mathcal{A}, v) \models Q_\tau x\big[\varphi_1(x), \ldots, \varphi_k(x)\big]$$

where $\mathcal{A}$ is a model (giving interpretations to all predicates, functions, and constants) and v is a valuation (a map from the set of variables to the domain A of $\mathcal{A}$; these are used as the values of the variables at their free occurrences). Denoting

$$v[x/a](y) = \begin{cases} a & \text{if } x = y, \\ v(y) & \text{otherwise,} \end{cases}$$

one can define for each formula $\varphi(x)$ a set

$$R^{\mathcal{A},x,v}_{\varphi(x)} = \left\{ a \in A \;\middle|\; (\mathcal{A}, v[x/a]) \models \varphi(x) \right\},$$

i.e., the set of values for x that render this formula satisfied. The rule for Q_τ is then that

$$(\mathcal{A}, v) \models Q_\tau x\big[\varphi_1(x), \ldots, \varphi_k(x)\big] \quad \text{iff} \quad \tau \ni \left(A, R^{\mathcal{A},x,v}_{\varphi_1(x)}, \ldots, R^{\mathcal{A},x,v}_{\varphi_k(x)}\right),$$

that is, τ is essentially a table of those combinations of domain elements and corresponding truth values for argument formulae that satisfy this quantifier. (The reason the domain A is included in the tuples of τ is that when the quantifier is defined in this generality, the "relational system" τ is required to be closed under isomorphism; if $(A, R_1, \ldots, R_k) \in \tau$ and $f\colon A \longrightarrow B$ is a bijection, then $\big(B, f(R_1), \ldots, f(R_k)\big) \in \tau$ as well.)

As an example of this, consider the *Härtig quantifier* Q_H for which the rule is

$$(\mathcal{A}, v) \models Q_H x\big[\varphi_1(x), \varphi_2(x)\big] \quad \text{iff} \quad \big|R^{\mathcal{A},x,v}_{\varphi_1(x)}\big| = \big|R^{\mathcal{A},x,v}_{\varphi_2(x)}\big|,$$

i.e., $Q_H x\big[\varphi_1(x), \varphi_2(x)\big]$ if and only if there are as many x satisfying $\varphi_1(x)$ as there are x satisfying $\varphi_2(x)$ (which is quite different from $\varphi_1(x)$ and $\varphi_2(x)$ being satisfied by the same x). As a relational system,

$$H = \left\{ (A, P, Q) \;\middle|\; P \subseteq A, Q \subseteq A, |P| = |Q| \right\}.$$

Another such quantifier is Q_{most}, where

$$most = \left\{ (A, P, Q) \;\middle|\; P \subseteq A, Q \subseteq A, |P \cap Q| > |P \setminus Q| \right\},$$

i.e., $Q_{most} x\big[\varphi_1(x), \varphi_2(x)\big]$ could be read as 'most x such that $\varphi_1(x)$ satisfy $\varphi_2(x)$' or in less formulaic language 'most φ_1 are φ_2' (compare 'most continuous functions are nowhere differentiable'). By [11, Th. 19.4], this Q_{most} quantifier is not definable in terms of any set of quantifiers of type $\langle 1 \rangle$ (i.e., acting upon only one subformula)! Hence it is not expressible using present OpenMath with less than that one introduces the primitives of set theory, which however throws all hope of something like algorithmic decidability out the window.

3 Standard Enhancement Proposal

As hinted at above, the PROPOSAL is that the restriction on a binding OpenMath object that it must have exactly one body subobject should be lifted. In the normative Relax NG schema for the OpenMath XML encoding, that corresponds to the change

```
--- openmath2.rng 2013-06-03 17:20:57.000000000 +0200
+++ openmath2+polybind.rng 2013-06-03 17:23:24.000000000 +0200
@@ -144,9 +144,11 @@
   <!-- binding constructor -->
   <define name="OMBIND">
     <element name="OMBIND">
       <ref name="compound.attributes"/>
       <ref name="omel"/>
       <ref name="OMBVAR"/>
-      <ref name="omel"/>
+      <zeroOrMore>
+        <ref name="omel"/>
+      </zeroOrMore>
     </element>
   </define>
```

It is not clear that there are any use-cases for the 'no bodies' edge case, but it should not hurt to allow it, and it could conceivably turn up as a base case for some sort of "n-ary binder".

Note that the proposed change, both for the XML encoding and for the binary encoding, is strictly a matter of relaxing the syntax; it does not require any new delimiters or separators for disambiguating. In the XML encoding, this is because the list of variables being bound is already bundled into one OMBVAR element. In the binary encoding, token identifiers 28 and 29 similarly delimit the sequence of variables being bound.

What would require new delimiters is however the informal function-style syntax used to define OpenMath objects in Chapter 2 of the standard, since this presently employs a flat list of head, variables, and body as in Subsection 2.1.3 Clause (iv):

$$\mathbf{binding}(B, v_1, \ldots, v_n, C)$$

It is thus further PROPOSED that the punctuation before and after the list of variables in a **binding** is changed to a semicolon, like so:

$$\mathbf{binding}(B; v_1, \ldots, v_n; C) \quad \text{and} \quad \mathbf{binding}(B; v_1, \ldots, v_n; C_1, \ldots, C_m)$$

This places the semicolons in the same positions as the tokens 28 and 29 in the binary encoding. The second semicolon is of course necessary to mark the boundary between the two variable-length sections. The first semicolon has the merit of highlighting the boundary between the head—where occurrences of the variables $v_1, \ldots, v_n$ are *not* bound by this binding—and the rest of the binding object—where these variables *are* bound by it (in the sense of participating in α-conversion). Since the latter is a somewhat subtle point, highlighting this boundary should enhance readability. Introducing semicolons as a new punctuation mark is furthermore not a very dramatic step, since these formulae already employ spaces as the secondary punctuation mark separating attribution symbols from their attribute value; commas never were the only punctuation marks around.

4 Anticipating Certain Objections

Since there has been a previous enhancement proposal [4] much to the same end as this one, it can be anticipated that objections raised back then will be put forth again. Hence there is a point in considering these objections and the problems with them.

4.1 The Dummy Helper Application

One line of reasoning holds that it is *unnecessary* to allow

$$\mathbf{binding}(B; v_1, \ldots, v_n; C_1, \ldots, C_m) \tag{4}$$

since the same information can be encoded as

$$\mathbf{binding}(B; v_1, \ldots, v_n; \mathbf{application}(D, C_1, \ldots, C_m))$$

where D is some new application symbol—a single new symbol could suffice for all binders. Proponents of this view might even think that (3) vindicates it, since the brackets there is a perfect match for this D application.

What the latter part of this argument overlooks is that the brackets in (3) are a purely notational device—much like the parenthesis in '$f(x, y, z)$'. Content-wise, the problem with the above argument is that it introduces into the formula an OpenMath object which is a purely syntactic device; a theory encompassing the generalised binder B will have interpretations for all of $C_1, C_2, \ldots, C_m$, and (4) as a whole, but there will not in general be anything that semantically corresponds to the $\mathbf{application}(D, C_1, \ldots, C_m)$ (even if there can be in specific cases). Hence this approach would prevent formulae from having a *native* expression in OpenMath, rather requiring them to be *encoded* into OpenMath (just like OpenMath in turn tends to be encoded into XML).

The dummy helper argument—that a multiplicity of objects can instead be encoded as a single application object—also begs the question of why **error** should be allowed to take an arbitrary number of non-heads, when **binding** is not. If minimality is the argument against generalised binders, then it should apply equally much against the present errors.

4.2 The Universal Lambda

Another line of reasoning holds that `fns1#lambda` is the only binder anyone really needs—that every other binder can be rewritten to a combination of lambda and an application—and moreover that doing so is The Future. Therefore any changes made in this area should rather be in the direction of deprecating all binders that are not `fns1#lambda`.

It is difficult to view this argument as being anything but primarily ideological: a statement of what one would *like* people to do, possibly for aesthetic reasons, rather than a conclusion founded in utility. It is quite true that binders are awkward to formalise (and I for one have much affinity for variable-free formalisms), but what is awkward about binders is the binding of variables as such rather than the concepts that the binder seeks to encode. Therefore the 'one lambda fits all' approach succeeds rather in combining the worst of both worlds: none of the conceptual unity that a specialised binder can bring, but all of the formal complications that are inherent in binding variables! Perhaps it is rationalised by viewing phrasebooks mostly as thin wrappers over an implementation language having lambda (but no other binders) as a primitive, since in that case having no other binders around makes life easier for the phrasebook implementor, but that would be a narrow-minded view. It cannot be the role of a general

standard such as OpenMath to say that some styles of doing math (e.g. the functional style) are good and others (e.g. using binders) are bad, even if individual phrasebooks of course may choose to support the symbols of one style but not those the other.

More down to earth, the 'one lambda fits all' approach to binding has several problems. One is that using a separate single-body lambda for each C_i of a generalised binding like (4) weakens the connection between occurrences of the same variable in different bodies, as remarked in [5]. Another problem is that it opens up for ontological contamination, both in that the use of the lambda suggests that all variable binding ends up substituting values for the variables, and conversely that the meaning of `fns1#lambda` as constructing an honest value-in-value-out applicable *function* is diluted by using it for random binding tasks. Third, the use of lambdas carries within it a presumption that functions are supported as first-class objects, even though first-class status of functions is (among formalisations of mathematics) a high-end feature rather then a basic one; domain-specific systems rather have the tendency to only accept known basic symbols as applications (unless the system happens to embed some variant of lambda calculus).

4.3 MathML Misalignment

A third line of reasoning could be that the proposed enhancement in principle is correct, but cannot be implemented because it is not in MathML 3. In other words, this amounts to saying "this can't be fixed, because the Other Standard is broken in this regard too."

However, since (Strict Content) MathML 3 only claims to be in alignment with OpenMath 2.0, there is no particular reason why OpenMath could not advance. To preserve interoperability, one could adapt a variation on the work-around from Subsection 4.1 to the conversion of generalised OpenMath bindings to Content MathML 3; this is well on par with other code transformations that such a conversion must perform, at least if viewed at the XML level.

A An Afterthought

While writing about the suggested modifications to the informal syntax for OpenMath **binding** objects, it occurs to me that this also opens up for unambiguously giving several subobjects *before* the list of variables. Could that be useful? Very much so, since it is hard to imagine a more straightforward encoding than

$$\mathbf{binding}\big(\textstyle\int, a, b; x; f(x)\big)$$

of the binding definite integral $\int_a^b f(x)\,dx$ than that! (With the physicist notation $\int_a^b dx\, f(x)$ the match is even closer.) The trick here is that since variables do not get bound before the first semicolon (i.e., before the OMBVAR child), that will be the correct place to include subobjects that should not be subjected to such binding, for example integral bounds. In the Relax NG schema, the corresponding change would be

```
@@ -144,9 +144,13 @@
   <!-- binding constructor -->
   <define name="OMBIND">
     <element name="OMBIND">
       <ref name="compound.attributes"/>
-      <ref name="omel"/>
+      <oneOrMore>
```

```
+        <ref name="omel"/>
+      </oneOrMore>
       <ref name="OMBVAR"/>
-      <ref name="omel"/>
+      <zeroOrMore>
+        <ref name="omel"/>
+      </zeroOrMore>
     </element>
   </define>
```

which is again an unambiguous relaxation.

The reason I do not up-front propose this slightly more general enhancement is that I have thought about it too late to be comfortable that I have given all consequences a reasonable consideration. It does for example seem plausible that it would be more disruptive to existing phrasebooks than the PROPOSED enhancement, since it should often have seemed natural to hardwire the knowledge that the OMBVAR is the *second* child of an OMBIND. The work-around by means of a helper **application** is also slightly less wasteful than for the bodies, since one can write

$$\mathbf{binding}\big(\mathbf{application}(\textstyle\int, a, b); x; f(x)\big)$$

and still get by with only one symbol (even if that is a weird application symbol in returning a binder, and moreover vulnerable to several of the arguments in Section 4).

This possibility is probably best discussed at the workshop.

B Binders in the Small Type System

Generalising the binders might also require a generalisation of their declarations in the Small Type System [3], but on the other hand that (at the time of writing) seems to be somewhat in disarray; all extant STS declarations of binders consist of the single symbol `sts#binder`, and do thus not match the `BIND` production in [3]. Moreover, the interpretation of that `BIND` production is unclear, since a full typing of even an ungeneralised binder should specify at least three pieces of information:

1. the types of the variables being bound,
2. the type of the body subobject, and
3. the type of the binding object as a whole.

However, the `BIND` production only has room for two pieces of information. Thus, something is missing anyway.

References

[1] Bruno Courcelle. Graph structure and monadic second-order logic: language theoretical aspects. *Lecture Notes in Comput. Sci.* **5125** (2008), 1–13.

[2] Bruno Courcelle and Joost Engelfriet. *Graph structure and monadic second-order logic.* Cambridge University Press, 2012. ISBN 978-0-511-97761-9. `http://www.labri.fr/perso/courcell/Book/TheBook.pdf`

[3] James Davenport. A small OpenMath type system. *Bulletin of the ACM Special Interest Group on Symbolic and Automated Mathematics (SIGSAM)* **34** no. 2 (2000), 16–21.

[4] James H. Davenport and Michael Kohlhase. Quantifiers and Big Operators in OpenMath. *22nd OpenMath Workshop*, 2009. http://kwarc.info/kohlhase/papers/om09-quantifiers.pdf

[5] James H. Davenport and Michael Kohlhase. Unifying Math Ontologies: A tale of two standards. Pp. 263–278 in *MKM/Calculemus Proceedings*, 2009. ISBN 978-3-642-02613-3. http://opus.bath.ac.uk/13079/

[6] Heinz-Dieter Ebbinghaus and Jörg Flum. *Finite model theory*. Springer, 1999. ISBN 3-540-65758-4.

[7] Donald E. Knuth. Two notes on notation. *Amer. Math. Monthly* **99**, no. 5 (1992), 403–422.

[8] Per Lindström. First-order logic. *Philosophical Communications* (ISSN 1652-0459), Web Series, No **36**. Göteborg : Department of Philosophy, Göteborg University, Sweden, 2006. http://www.phil.gu.se/posters/first-order-logic.pdf

[9] William R. Parzynski and Philip W. Zipse. *Introduction to mathematical analysis*. McGraw-Hill, 1982.

[10] "Quantifier", entry in *Encyclopaedia of mathematics*. http://eom.springer.de/Q/q076260.htm

[11] Dag Westerståhl. "Quantifiers", Chapter 19 (pp. 437–460) in: Lou Goble. *The Blackwell Guide to Philosophical Logic*. Blackwell Publishing, 2001.